Our Ref.: 41007.P008

APPLICATION FOR UNITED STATES LETTERS PATENT

FOR

Network Traffic Data Collection and Query

Inventor(s):
Christopher L. Cook
Gretta E. Bartels

Prepared by:
Columbia IP Law Group, PC

"Express Mail" label number EL743035479US

Network Traffic Data Collection and Query

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of networking. More specifically, the present invention relates to the collection of network traffic data, and the selective analysis of the collected data.

2. Background Information

With advances in integrated circuit, microprocessor, networking and communication technologies, an increasing number of devices, in particular, digital computing devices, are being networked together. Devices are often first coupled to a local area network, such as an Ethernet based office/home network. In turn, the local area networks are interconnected together through wide area networks, such as SONET networks, ATM networks, Frame Relays, and the like. Of particular notoriety is the TCP/IP based global inter-networks, Internet.

As a result this trend of increased connectivity, an increasing number of applications that are network dependent are being deployed. Examples of these network dependent applications include but are not limited to, email, net based telephony, world wide web and various types of e-commerce. For these applications, success inherently means high volume of network traffic for their implementing servers. To ensure continuing success, quality of service through orderly and efficient handling of the large volume of network traffic has become of paramount importance. Various subject matters, such as scalability, distributive

deployment and caching of contents as well as preventing network misuse have become of great interest to the artisan.

Many of these subject matters rely on the collection and analysis of network traffic data. A number of prior art tools are available to perform such data collection and analyses. An example of such prior art tool is the NetFlow Collector and Analyzer available from CISCO, Inc. of San Jose, CA. These early generation data collection and analysis tools all suffer from at least a common deficiency in that they lack a flexible and versatile query facility that allows a wide variety of interrogatories to be made against a collection of network traffic data, to facilitate in-depth analysis of the network traffic. Such in-depth analyses are increasingly needed to understand and manage the performance of today and future complex networks.

Thus, a more flexible and versatile methodology is desired.

SUMMARY OF THE INVENTION

An apparatus is equipped to receive descriptive data for network traffic. In one embodiment, the apparatus is equipped to conditionally modify timing data of the network traffic to conform the timing data to the timing patterns of previously network traffic, when determined that the timing data of the network traffic are aberrations. Further, the apparatus is equipped with a query facility that supports a network oriented query language. The language includes specific network oriented language elements.

In one embodiment, the language elements include

a) command options for specifying a query to be run for a plurality of time bins;

b) commands for specifying processing to be performed for each network traffic flow;

c) commands for specifying processing to be performed for a first, a last or each packet; and

d) predetermined network oriented keywords for specifying network traffic attributes.

BRIEF DESCRIPTION OF DRAWINGS

The present invention will be described by way of exemplary embodiments, but not limitations, illustrated in the accompanying drawings in which like references denote similar elements, and in which:

Figure 1 illustrates an overview of the present invention, in accordance with one embodiment;

Figure 2 illustrates a method view of the data collection aspect of the present invention, in accordance with one embodiment;

Figure 3 illustrates a data format for storing the collected network traffic data in temporary storage, in accordance with one embodiment;

Figure 4 illustrates a data format for storing the collected network traffic data in persistent storage, in accordance with one embodiment;

Figures 5a-5b illustrate two panes of an end user interface for configuring and managing the data collection and query facility of the present invention, in accordance with one embodiment;

Figures 5c-5d illustrate two panes of the end user interface of the query facility of the present invention, in accordance with one embodiment;

Figures 6a-6b illustrate the query command feature of the query facility of the present invention, in accordance with one embodiment;

Figure 7 illustrates the operational flow of the relevant aspects of the query compiler of **Fig. 1**, in accordance with one embodiment; and

Figure 8 illustrates the operational flow of the relevant aspects of the query execution engine of **Fig. 1**, in accordance with one embodiment;

Figure 9 illustrates an example computer system suitable for use to host a software implementation of the data collection and query facilities of the present invention, in accordance with one embodiment; and

Figures 10a-10e illustrate one example implementation of the byte code generation.

GLOSSARY

ATM	Asynchronous Transfer Mode
AS	Autonomous System
DNS	Domain Name Service
DSCP	Differentiated Services Code Point
IP	Internet Protocol
SONET	Synchronous Optical Network
TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol
Unixsecs	Time in seconds

DETAILED DESCRIPTION OF THE INVENTION

In the following description, various aspects of the present invention will be described. However, it will be apparent to those skilled in the art that the present invention may be practiced with only some or all aspects of the present invention. For purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will also be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known features are omitted or simplified in order not to obscure the present invention.

Parts of the description will be presented in terms of operations performed by a processor based device, using terms such as receiving, analyzing, determining, generating, and the like, consistent with the manner commonly employed by those skilled in the art to convey the substance of their work to others skilled in the art. As well understood by those skilled in the art, the quantities take the form of electrical, magnetic, or optical signals capable of being stored, transferred, combined, and otherwise manipulated through mechanical and electrical components of the processor based device; and the term processor include microprocessors, micro-controllers, digital signal processors, and the like, that are standalone, adjunct or embedded.

Various operations will be described as multiple discrete steps in turn, in a manner that is most helpful in understanding the present invention, however, the order of description should not be construed as to imply that these operations are necessarily order dependent. The terms "routing devices" and "route" are used throughout this application, in the claims as well as in the specification. The terms as

used herein are intended to be genus terms that include the conventional routers and conventional routing, as well as all other variations of network trafficking, such as, switches or switching, gateways, hubs and the like. In particular, these operations need not be performed in the order of presentation. Further, the description repeatedly uses the phrase "in one embodiment", which ordinarily does not refer to the same embodiment, although it may.

Overview

Referring now to **Figure 1**, wherein a block diagram illustrating an overview of the present invention, in accordance with one embodiment, is shown. As illustrated, the present invention includes network traffic data collection facility **101** and network traffic data query facility **103**. Network traffic data collection facility **101** includes network traffic data collector **102**, associated temporary data storage buffer **104**, and associated persistent data storage **106**. Network traffic data query facility **103** includes end user interface **108**, query generator **110**, query compiler **112** and query execution engine **114**. The elements are operationally coupled to each other as shown.

Data collector **102** is provided to facilitate collection of descriptive data associated with network traffic of interest. As will be described in more detail below, data collector **102** advantageously includes "auto timing correction" for the descriptive data being collected. That is, data collector **102** detects timing "aberrations", and automatically corrects timing data that are deemed to be "aberrations". Timing "aberration" refers to the phenomenon where a significant difference exists between a routing device and its "monitoring" sensing device on the perception of time, and/or the timing data for network traffic between two nodes exhibits a temporal or sudden substantial difference from timing patterns of early

network traffics between the same two nodes or two nodes of the same communities. Such timing "aberrations" are known to exist between routing devices and their "monitoring" sensing devices, as well as known to occur from time to time for network traffic packets routed on a "best effort" basis (such as, network traffic packets routed over the Internet).

In one embodiment, the descriptive data are collected for network traffic flows. A network traffic flow comprises source address, destination address, source port, destination port, Layer 3 protocol, TOS byte (DSCP), and identification of the ingress interface.

Temporal data buffer **104** and persistent storage **106** are provided to store the collected network traffic data. The network traffic data are typically first stored in temporal data buffer **104** before being stored into persistent storage **106**. However, in the presently preferred embodiments, direct storage into persistent storage **106** is also provided. Further, network traffic data stored in persistent storage **106** may be reloaded into temporal data buffer **104** for processing. In alternate embodiments, queries may be run directly against network traffic data stored in persistent storage **106**, without first explicitly reloading them back into temporal data buffer **104**. In one embodiment, temporal data buffer **104** is a circular buffer disposed in system memory of the "host" system "hosting" data collector **102**, and persistent storage **106** is disposed on a mass storage device, such as a disk drive, of the "host" system (not shown) of data collector **102**. In alternate embodiments, persistent storage **106** may be disposed on a distributed storage device, such as network storage, accessible from the "host" system.

End user interface **108**, as will be described in more detail below, is provided to facilitate a user in submitting "high level" (abstracted) query commands against the collected network traffic data. These query commands result in queries formed

using language elements of the network oriented query language of the present invention, to be run against the collected network traffic data. In one embodiment, end user interface **108** also includes features that allow an administrator having the proper authority to configure data collection facility **101** and query facility **103**. In alternate embodiments, separate interfaces may be used for the administration functions instead.

Query generator **110**, as will be described in more detail below, is provided to process the "high level" (abstracted) query commands, and in response, "generates" the appropriate query or queries formed with language elements of the network oriented query language of the present invention.

Query compiler **112**, as will be described in more detail below, is provided to "compile" queries formed using language elements of the network oriented query language of the present invention, into byte codes for execution under the control of query execution engine **114**. Query execution engine **114**, as will be described in more detail below, control execution of the byte codes as well as performs conventional execution runtime services, such as memory allocation request and release, error/exception handling, and so forth.

For the illustrated embodiment, query compiler **112** includes an application programming interface (API) (not shown), through which queries formed using language elements of the network oriented query language of the present invention, may be programmatically submitted by other processes, such as network management processes **116**, for compilation, and in turn execution by engine **114**. Management processes **116** may be co-resident with query facility **103** on the same "host" system (not shown) of query facility **103** or remotely disposed away the "host" system, and communicate with query facility **103** using conventional cross system communication techniques. In alternate embodiments, the various components of

query facility **103**, i.e. end user interface **108**, query generator **110**, and so forth, may be co-resident on the same "host" system, or distributedly disposed. Again, the distributedly disposed components may communicate with each other using conventional cross system communication techniques.

The relevant aspects of the various elements, including the manner they cooperate with one another will be described in turn below.

Data Collection

Referring now also to **Figure 2**, wherein a flow chart illustrating the operational flow of the relevant aspects of data collector **102**, in accordance with one embodiment, is shown. As illustrated, at block **202**, data collector **102** receives a "reporting" of descriptive data associated with certain network traffic of interest. The descriptive data may be provided by a routing device routing network traffic, a sensor sensing or monitoring network traffic being routed, or other devices of the like. The provision may be made as part of a periodic reporting the routing or sensor device is configured to make periodically, or the provision may be made in response to an inquiry by data collector **102**, or at the direction of a "director" device directing distributed network traffic management. One example of such "director" device is described in co-pending U.S. patent applications, number 09/631,898, entitled "A Distributed Solution For Regulating Network Traffic", filed on August 4, 2000, and number 09/685,518, entitled "Progressive and Distributed Regulation of Selected Network Traffic Destined for a Network Node", filed on October 9, 2000. These applications are hereby fully incorporated by reference.

For the illustrated embodiment, data collector **102** supports a plurality of formats under which the descriptive data may be reported. Accordingly, at block **204**, data collector **102**, upon receipt of the reported data, first determines the format

of the descriptive data. It is assumed each support format may be uniquely determined. For example, each format may include an identifier or an identifying attribute disposed at certain position of the data. For the multi-format embodiment, if the data format is not discernable, the data is discarded, and an error is logged (not shown).

Upon determining the data format of the received data, data collector **102**, among other things, proceeds to analyze the timing data. As alluded to earlier, a routing device's perception of time may be significantly different from that of its sensors. Additionally, for packet traffic being routed on a "best effort" basis over a network such as the Internet, from time to time, the timing data, i.e. the time elapsed from the time a packet is sent out of a source node to the time the packet arrives at a data node may become distorted. To further illustrate, when packet is sent from node A to node B, most of the times the timing data would reflect a 10 +/- 1 "timing ticks" as transmit time, however from time to time, unusually large timing values, like 50 or unusually small timing values, like 1, may be reported. In these cases, data collector **102** advantageously recognizes these unusually large or unusually small timing values as "aberrations", and automatically corrects them accordingly.

Thus, for the embodiment, at block **208**, data collector **102** determines if the timing data of the reported data is "consistent" with the timing patterns of other routing devices and/or prior network traffic. Data collector **102** maintains profile records of the historic timing patterns. If data collector **102** determines that the timing data are not consistent with the timing patterns of other routing devices and/or prior network traffic, at block **210**, data collector automatically adjusts the timing data to conform the timing data of the received network traffic to the timing patterns of the other routing devices and/or prior network traffic. The amount of modification to be applied to adjust the timing data to conform to the timing patterns of other routing

devices and/or prior network traffic data is application dependent, and preferably configurable by an administrator with the proper authority.

At block **212**, either upon determining that the timing data is consistent or the making the adjustment, data collector **102** saves the descriptive data into the temporal and/or persistent storage **104-106**. For the embodiment, data collector **212** also converts and saves the received data in a common format to facilitate more efficient operation for subsequent query processing. Further, at block **214**, data collector **214** update the adjustment data it employs to "auto correct" the timing data, based on the timing data of the received network traffic data.

In one embodiment, data collector **102** employs a pair of deviation thresholds +/- "delta" to determine whether the timing data of the received data are to be considered as aberrations or not. In one embodiment, "delta" is one standard deviation of the earlier timing data. In one embodiment, the adjustment to be applied in an "auto correct" situation is an adjustment value necessary to bring the timing data of the received data into alignment (e.g. within the earlier mentioned +/- "delta" threshold range) with a weighted running average of the timing data. In one embodiment, the weighting favors the historic data over the most recent data. That is, the weighting scheme implicitly gives more credence to earlier observed timing patterns then to the most recently observed timing behavior.

Figure 3 illustrates the common data format for storing the network traffic data in temporal data buffer **104**. As illustrated, the header values are stored in accordance with header format **302** (32 bytes for the embodiment), whereas the entry values are stored in accordance with entry format **304** (52 bytes for the embodiment). In alternate embodiments, other data formats may be practiced instead.

Similarly, **Figure 4** illustrates the common data format for storing the network traffic data in persistent data storage **106**. As illustrated, the header values are stored in accordance with file header format **402** (16 bytes for the embodiment), whereas the entry values are stored in accordance with data header format **404** (28 bytes for the embodiment).

Management/Configuration

Referring now to **Figures 5a-5b**, wherein two block diagrams illustrating two panes of end user interface **108** for use to manage or configure the data collection and query facilities of the present invention, in accordance with one embodiment, are shown. As alluded to earlier, these panes, panes **502** and **512** are made available to an administrator with proper authority. Enforcement or ensuring the administrator as one having the necessary authority, may be accomplished using anyone of a number of techniques known in the art, such as through the "log in" process of the "host" system.

As illustrated, pane **502** includes drop list **504** for presenting the configurable parameters, and field **506** for setting the parameter value of the selected parameter. In one embodiment, the parameters include, but are not limited to

Name of Field	Type	Description	Default Value
BufferSize	Number	Temporal buffer size in bytes	1M
RecvQueueSize	Number, or MAX	Kernel buffer size in bytes	MAX
UdpPorts	Numbers, space delimited	Which UDP ports to listen on for network traffic flow data	2055
TcpPorts	Numbers	Which TCP ports to listen on for	2001

	space delimited	client connections	
Monitoring	ON or OFF	When OFF, incoming network traffic flow is dropped (rather than being added to the temporal buffer)	ON
Logging	ON or OFF	When ON, incoming network traffic flow is written into a system log file	OFF
Checksumming	ON or OFF	When ON, checksum is performed on input files.	ON
AllowedUdpSenders	Space delimited IP addrs, or ALL, or NONE	Which IP addresses may send network traffic flow data	ALL
AllowedTcpClients	Space delimited IP addrs, or ALL, or NONE	Which IP addresses may connect as clients	ALL
MaxLogFileSize	Number	After reaching this size in bytes, close the current log file and start a new one	100M
RecycleLogFiles	ON or OFF	If ON, when disk fills up, replace oldest log file; if OFF, just stop logging	OFF
FlushDelay or LogFileFlushDelay	Number	Max seconds before logged data is flushed to disk	5

Pane **512** is provided to facilitate an authorized administrator in performing various management operations, including state and/or status checking. Pane **512** includes in particular field **514** for entering the various management operation commands. In one embodiment, the management commands include, but are not limited to

Command	Description
show buffersize	Show the size in bytes of the temporal data buffer
show datasize	Show the size in bytes of the data present in the temporal buffer
show file logging	Show whether file logging is enabled
show file recycling	Show whether file recycling is enabled
show monitoring	Show whether monitoring is enabled
show checksumming	Show whether checksumming is enabled
show ports	Show the UDP ports being listens for network traffic flow
show query version	Show the current version of the query language
show current time	Show the current time in UnixSecs

Command	Effect
set buffersize <number>	Set the size in bytes of the temporal data buffer
set maxlogfilesize <number>	Set the size in bytes of the maximum log file
start monitoring	Start monitoring network traffic flow
stop monitoring	Stop monitoring network traffic flow
start file logging	Start file logging
stop file logging	Stop file logging
start file recycling	Start file recycling
stop file recycling	Stop file recycling
set checksumming {ON/OFF}	Turn checksumming on or off
add port <number>	Add a UDP port to ports to be listen
remove port <number>	Remove a UDP port from ports to be listen

Command	Effect
save to <filename>	Save the data in the temporal buffer to specified file

load from <filename>	Load the data in the specified file into the temporal buffer

Referring now to **Figures 5c-5d**, wherein two block diagrams illustrating two panes of end user interface **108** for use to run queries against a collection of network traffic data, in accordance with one embodiment, are shown. As illustrated, panes **522** and **532** include fields **524** and **534** for a user to submit "high level" (abstracted) queries that result in queries formed with language elements of the network oriented query language of the present invention or "advanced" queries expressed with the language elements.

As those skilled in the art would appreciate, panes **502-532** in practice typically include many more end user interface features, such as buttons and the like. These features are not illustrated, so not to obscure the present invention.

<u>Query Generation</u>

Referring now to **Figures 6a-6b**, wherein two block diagrams illustrating query generator **110** of **Fig. 1** in further details, in accordance with one embodiment, are shown. As illustrated in **Fig. 6a**, query generator **110** includes library **604** of parameterized pre-packaged queries formed with language elements of the network oriented query language of the present invention, and logic **602** for looking up the parameterized pre-packaged queries based on "high level" (abstracted) query "commands".

As illustrated in **Fig. 6b**, upon invocation, at block **612**, query generator **110** extracts the "keywords" from the "high level" (abstracted) query "command" entered. At block **614**, query generator **110**, using the extracted "keyword", looks up the corresponding pre-packaged query. At block **616**, query generator **110** resolves the

query parameters of the retrieved pre-packaged query, based on the "parameter values" submitted in conjunction with the "high level" (abstracted) query "command" entered. Upon processing the parameters, query generator **110**, at block **618** outputs the pre-packaged query for compilation by query compiler **112**, and eventual execution under the control of execution engine **114**.

In one embodiment, the "high level" (abstracted) query "commands" and their corresponding queries include

Query Command	Query

mq hist keys values conditions	mquery {
	DefHist 1 Keys = keys Values = values
	If DataPresent ForEachFlow {
	If conditions {
	AddToHist 1
	}
	}
	PrintHist 1
	}

mq top100 keys *conditions*	mquery {
	DefArray 1 keys
	If DataPresent ForEachFlow {
	If conditions {
	AddToArray 1
	}
	}
	command to print top100 output
	}

mq hdrhist keys *values conditions*	mquery {
	DefHist 1 Keys = keys Values = values
	If DataPresent ForEachPacket {
	If conditions {
	If (var255 == 0)
	var255 = BaseSeqNumber
	AddToHist 1
	Set var255 = BaseSeqNumber + NumEntriesInPacket
	}

	} PrintHist 1 }

Listed above are just examples of some of the "high level" (abstracted) query commands that can be supported under the present invention. The meaning of the various language elements are either self-explanatory or will be readily apparent in view of the descriptions to follow. As those skilled in the art will appreciate, especially in view of the descriptions to follow, the present invention may be practiced with many more "high level" (abstracted) query commands (or less).

<u>Network Oriented Query Language</u>

In accordance with the presently preferred embodiment, the network oriented query language of the present invention includes the language elements, and syntax as set forth below:

Language Element	**Expansion**
<MQuery>	**mquery** <CommandBlock> | **mbinnedquery** <CommandBlock> *binsize* | **mbinnedquery** <CommandBlock> *binsize start stop* *Binsize, start, and stop are in seconds; Start and stop are in the UnixSecs domain while binsize is an absolute number*
<CommandBlock>	<Command> | { <CommandList> }
<CommandList>	<Command> | <Command> <CommandList> *Commands may be separated by newlines or semicolons if you wish but it is not required.*
<Command>	<IfCommand> | <SetCommand> | <IncrCommand> |

<IncrByCommand> |
<DefHistCommand> |
<DefArrayCommand> |
<WithFirstPacketCommand> |
<WithLastPacketCommand> |
<ForEachPacketCommand> |
<ForEachFlowCommand> |
<AddToHistCommand> |
<AddToArrayCommand> |
<PrintCommand> |
<NewLineCommand> |
<PrintHistCommand> |
<PrintArrayCommand>

<IfCommand>
If <Expr> <CommandBlock> |
If <Expr> <CommandBlock> **Else** <CommandBlock>

<SetCommand>
Set <Var> <Expr> |
Set <Var> **=** <Expr> |
<Var> **=** <Expr>

<IncrCommand>
Incr <Var> |
<Var>**++**

<IncrByCommand>
Incr <Var> **By** <Expr> |
<Var> **+=** <Expr>

<DefHistCommand>
<DefHist> <Index> <KeyExpr> |
<DefHist> <Index> <KeyExpr> <ValExpr>

<DefArrayCommand>
Define Array <Index> <Expr> |
DefArray <Index> <Expr>

<WithFirstPacketCommand>
<WithFirstPacket> <CommandBlock>

<WithLastPacketCommand>
<WithLastPacket> <CommandBlock>

<ForEachPacketCommand>
<ForEachPacket> <CommandBlock>

<ForEachFlowCommand>
<ForEachFlow> <CommandBlock>

<AddToHistCommand>
Add to Histogram <Index> |
Add to Hist <Index> |
AddToHist <Index> |
AddHist <Index>

<AddToArrayCommand>	**Add To Array** <Index> | **AddToArray** <Index> | **AddArray** <Index>
<PrintCommand>	**Print "*string*"** | **Print_ "*string*"** | **Print** <Expr> | **Print_** <Expr> | **Print** <Lvar>
<NewLineCommand>	**NewLine** | **NL**
<PrintHistCommand>	**Print Histogram** <Index> | **Print Hist** <Index> | **PrintHist** <Index>
<PrintArrayCommand>	**Print Array** <Index> | **PrintArray** <Index>
<Expr>	**CurrentTime** | **DataPresent** | **MulDiv32** <Expr>**,** <Expr>**,** <Expr>| **MulDiv64** <Expr>**,** <Lvar>**,** <Lvar>| *32-bit constant integer* | *network address in dotted quad format* | <Var> | <HeaderValue> | <EntryValue> | <ExprLogical> | <ExprBitwise> | <ExprArithmetic> | <PrintType>**:**<Expr> | (<Expr>)
<Var>	**Var**<Index>
<DefHist>	**Define Histogram** | **Define Hist** | **DefHist**
<Index>	*A number from 0 to 255*
<KeyExpr>	**Keys** <ExprList> | **Keys =** <ExprList> | **Key** <ExprList> |

	Key = <ExprList> | **Bins** <ExprList> | **Bins =** <ExprList> | **Bin** <ExprList> | **Bin =** <ExprList>
<ValExpr>	**Values =** <ValueList>
<WithFirstPacket>	**With First Packet** | **WithFirst Packet** | **WithFirstPacket** | **For First Packet** | **ForFirst Packet** | **ForFirstPacket**
<WithLastPacket>	**With Last Packet** | **WithLast Packet** | **WithLastPacket** | **For Last Packet** | **ForLast Packet** | **ForLastPacket**
<ForEachPacket>	**For Each Packet** | **ForEach Packet** | **ForEachPacket**
<ForEachFlow>	**For Each Flow** | **ForEach Flow** | **ForEachFlow**
<Lvar>	**LVar**<Index>
<HeaderValue>	**OrigFlowType** | **NumEntriesInPacket** | **RouterUptime** | **UnixSecs** | **SensorUnixSecs** | **BaseSequenceNumber** | **EngineType** | **EngineId** | **UnixMSecs** | **AggregationMethod** | **AggregationVersion** | **SamplingInterval** | **SenderAddr**

<EntryValue>

SourceAddr |
DestAddr |
NextHop |
InInterface |
OutInterface |
NumPackets |
NumBytes |
FlowStartTime |
FlowEndTime |
SourcePort |
DestPort |
PAD8 |
TcpFlags |
Protocol |
TOS |
SourceAS |
DestAS |
SourceNetmask |
DestNetmask |
PAD16 |
NumFlows

<ExprLogical>

! <Expr> |
<Expr> **&&** <Expr> |
<Expr> **||** <Expr> |
<Expr> **<>** <Expr> |
<Expr> **!=** <Expr> |
<Expr> **==** <Expr> |
<Expr> **=** <Expr> |
<Expr> **<=** <Expr> |
<Expr> **<** <Expr> |
<Expr> **>=** <Expr> |
<Expr> **>** <Expr>

<ExprBitwise>

~ <Expr> |
<Expr> **&** <Expr> |
<Expr> **AND** <Expr> |
<Expr> **|** <Expr> |
<Expr> **OR** <Expr> |
<Expr> **^** <Expr> |
<Expr> **EOR** <Expr> |
<Expr> **XOR** <Expr>

<ExprArithmetic>

<Expr> **+** <Expr> |
<Expr> **-** <Expr> |

	<Expr> * <Expr> | <Expr> / <Expr> | <Expr> % <Expr> | <Expr> # <Expr>
<ExprList>	<Expr> | { <Expr> <Expr> ... <Expr> } *Multiple exprs in a list should be space-delimited*
<ValueList>	<Value> | { <Value> <Value> ... <Value> } *Multiple values in a list should be space-delimited*
<Value>	**Sum:**<Expr> | **Or:**<Expr> | **Min:**<Expr> | **Max:**<Expr> | **First:**<Expr> | **Last:**<Expr> | **Unique:**<Expr>
<PrintType>	**UInt:** | **Int:** | **CUInt:** | **CInt:** | **Secs:** | **MSecs:** | **TcpFlags:** | **Protocol:** | **IPAddr:** | **Bits:** | **Hex:**

The above example language elements support the singular execution of a query (mquery), and repeated execution of a query over a plurality of time bins (mbinnedquery). A time bin is synonymous with a time period. An example of a time bin is the time period from time unit 9:01 to 9:02. An example of a timing bin specification is (900, 1000, 10), which is interpreted to mean that the first time bin is to start at "time tick" 900, and the last time bin is to stop at "time tick" 1000, with

each time bin having a bin size of 10 "time ticks". Accordingly, the example time bin specification results in 10 time bins (over which the query is to be run).

The above example language elements assume two types of working data structures for returning the result of the query, a histogram data structure and an array data structure. The two data structures are created using the command element "Define Histogram" (also specifiable as Define Hist or DefHist), and "Define Array" (also specifiable as DefArray).

The query results are added to the histogram or array data structures through the corresponding "Add to" language elements. The query results are selected using the remaining language elements, which are substantially self-explanatory to those skilled in art. In particular, those skilled in the art will appreciate that inclusion of network oriented language elements such as the various versions of "With First Packet", "With Last Packet", "For First Packet", "For Last Packet", "For Each Packet", and the various predetermined network oriented "header value" and "entry value" keywords, renders the query language especially efficient in articulating interrogatories against a collection of network traffic data. Those skilled in the art will also appreciate that the present invention may be practiced with more or less language elements.

Query Compilation

Referring now to **Figures 7**, wherein a flow chart illustrating the operational flow of the relevant aspects of query compiler **112**, in accordance with one embodiment, is shown. As illustrated, upon invocation, at block **702**, query compiler **112** locates the "next" command of the query being compiled. Locating the "next " command of the query being compiled may be effectuated using any one of a number of "parsing" techniques known in the art. Upon locating the "next" command of the query being

compiled, at block **704**, query compiler **114** identifies the command read (e.g. from a language dictionary maintained by query compiler **112**). Next, at block **706**, query compiler **114** reads the rest of the command syntax, and determines the semantic for the identified command.

Then, at block **708**, query compiler **114** determines if end of query has been reached. If not, query compiler **114** returns to block **702**, and continues the compilation process from there. Eventually, the end of query being compiled is reached. At such time, query compiler **114** may optionally apply one or more optimizations to the commands analyzed, block **710**. The optimizations may be any one or more of the applicable compiler optimization techniques known in the art.

Finally, with or without optimizing the commands analyzed, at block **712**, query compiler **114** generates byte codes for the commands analyzed. **Figures 10a-10e** illustrate one example implementation of the byte code generation part of query compiler **114**. In alternate embodiments, other equivalent "code generation" approaches may be practiced instead. In one embodiment, query compiler **114** includes the size of the byte code generated for the query in the beginning of the executable byte codes.

Query Execution

Referring now to **Figures 8**, wherein a flow chart illustrating the operational flow of the relevant aspects of query execution engine **114** in accordance with one embodiment, is shown. As illustrated, upon invocation, if necessary (e.g. when invoked the first time), query execution engine **114** initializes the runtime environment for query execution, block **802**. Initialization tasks include but are not

limited to obtaining and initializing a memory pool allocation, loading an exception handler, and so forth.

Upon setting up the runtime environment, query execution engine **114** determines if the query to be executed is a binned query, block **803**. If so, query execution engine **114** further determines the time bin structure, i.e. the time bins over which the query is to executed.

Thereafter, query execution engine **114** successively fetches and dispatches the "byte codes" of the query to be executed, blocks **804-808**. As each "byte code" is executed, query execution engine **114** repeats the process, until eventually all generated "byte codes" of a query has been executed, for each and every applicable time bin. Execution of each byte code is dependent on the semantic of the corresponding command or commands, similar to other "byte code" based execution known in the art. These corresponding portions may be implemented in any one of a number of implementation approaches of these like kind "byte code" based execution facilities known in the art.

In one embodiment, as part of controlling the execution, query execution engine **114** automatically "apportions" the selected data between two time bins, when the selected data straddles two time bins and the query is a binned query. Query execution engine **114** analyzes the amount of "straddling" in each of the neighboring time bins, and proportionally apportions the data accordingly. In alternate embodiment, disproportionate or weighted apportionment may also be supported.

In one embodiment, as part of controlling the execution, query execution engine **114** also automatically amplifies the selected data, if the selected data was collected by the reporting/monitored device on a sampling basis. For the illustrated embodiment, query execution engine **114** also analyzes the data to determine the

sampling ratio, and amplifies accordingly. For example, in querying for packet count during a time period, upon determining that the packet count data was collected on a "1 of 10" sampling method, query execution engine **114** automatically amplifies the packet count data by 10x. In alternate embodiments, non-corresponding, i.e. amplification that is larger or smaller than the sampling ratio, may also be supported.

At the end of execution, for the illustrated embodiment, query execution engine **114** "tears down" the execution environment, block **810**. In alternate embodiments, fully or partially persistent runtime environment for different queries may be practiced instead.

Example Host Computer System

Figure 9 illustrates an example computer system suitable for use to be programmed with the data collection and query facilities **101** and **103** of the present invention, in accordance with one embodiment. As shown, computer system **900** includes one or more processors **902** (typically depending on whether it is used as host to sensor or the director), and system memory **904**. Additionally, computer system **900** includes mass storage devices **906** (such as diskette, hard drive, CDROM and so forth), input/output devices **908** (such as keyboard, cursor control and so forth) and communication interfaces **910** (such as network interface cards, modems and so forth). The elements are coupled to each other via system bus **912**, which represents one or more buses. In the case of multiple buses, they are bridged by one or more bus bridges (not shown). Each of these elements performs its conventional functions known in the art. In particular, system memory **904** and mass storage **906** are employed to store a working copy and a permanent copy of the programming instructions implementing the data collection and query facility of the present invention. The permanent copy of the programming instructions may be

loaded into mass storage **906** in the factory, or in the field, as described earlier, through a distribution medium (not shown) or through communication interface **910** (from a distribution server (not shown). The constitution of these elements **902-912** are known, and accordingly will not be further described.

Conclusion and Epilogue

Thus, it can be seen from the above descriptions, a novel method and apparatus for collecting network traffic data and querying the collected data has been described. The novel scheme is particularly useful in detecting misuse of network links, e.g. a denial of service attack.

While the present invention has been described referencing the illustrated and above enumerated embodiments, the present invention is not limited to these described embodiments. Numerous modification and alterations may be made, consistent with the scope of the present invention as set forth in the claims to follow. Thus, the above described embodiments are merely illustrative, and not restrictive on the present invention.
